

Overlapping Communities in Complex Networks



Jan Dreier

Bachelor's Thesis

Theoretical Computer Science
Department of Computer Science
RWTH Aachen University

2014

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, September 29, 2014
Jan Dreier

Contents

1	Abstract	5
1.1	English	5
1.2	Deutsch	6
2	Introduction	7
2.1	Notation	7
3	Previous Work	8
3.1	Non-Overlapping Community Detection	8
3.2	Overlapping Community Detection	9
4	Fundamentals	11
4.1	Markov Chains	11
4.2	Absorbing Markov Chains	13
4.3	Random Walks	14
4.4	Symmetric Diagonally Dominant Linear Systems	15
5	Our Model for Community Detection	17
5.1	Single Community Detection	19
5.2	Algorithm	20
5.3	Multiple Overlapping Community Detection	22
6	Experimental Results	24
6.1	Implementation	24
6.2	Benchmark	24
6.3	Evaluation	26
7	Conclusion	29

List of Figures

1	Graph representation of a Markov chain	12
2	Transition matrix of the same Markov chain	12
3	Comparison of random walk distance and shortest path distance in a graph.	18
4	Removal of outgoing edges of seed nodes in a graph	19
5	An example graph generated by the LFR benchmark.	26
6	Results of the benchmark	27
7	Summary of all parameters for the benchmark	27
8	Histogram over the distribution of Q	28

1 Abstract

1.1 English

Communities are subsets of a network that are densely connected inside and share only few connections to the rest of the network. The aim of this research is the development and evaluation of an efficient algorithm for detection of overlapping, fuzzy communities.

The algorithm gets as input some members of each community that we aim to discover. We call these members seed nodes. The algorithm then propagates this information by using random walks that start at non-seed nodes and end as they reach a seed node. The probability that a random walk starting at a non-seed node v ends at a seed node s is then equated with the probability that v belongs to the communities of s .

The algorithm runs in time $\tilde{O}(l \cdot m \cdot \log n)$, where l is the number of communities to detect, m is the number of edges, n is the number of nodes. The \tilde{O} -notation hides a factor of at most $(\log \log n)^2$.

The LFR benchmark proposed by Lancichinetti et al. is used to evaluate the performance of the algorithm. We found that, given a good set of seed nodes, it is able to reconstruct the communities of a network in a meaningful manner.

1.2 Deutsch

Eine Community ist eine Untermenge eines Netzwerks, welche intern stark verknüpft ist, jedoch nur wenige Verbindungen zum Rest des Netzwerks besitzt. Das Ziel dieser Arbeit ist die Entwicklung und Beurteilung eines effizienten Algorithmus zur Entdeckung von überlappenden, weichen Communitys.

Der Algorithmus erhält als Eingabe Mitglieder jeder Community, die er entdecken soll. Diese Mitglieder nennen wir Seed Knoten. Der Algorithmus verbreitet diese Information mit Hilfe von Random Walks, welche bei nicht-Seed Knoten anfangen, und enden sobald sie einen Seed Knoten erreichen. Die Wahrscheinlichkeit, dass ein Random Walk, der bei einem nicht-Seed Knoten v anfängt, bei einem Seed Knoten s endet, wird gleichgesetzt mit der Wahrscheinlichkeit, dass v den Communitys von s angehört.

Der Algorithmus läuft in Zeit $\tilde{O}(l \cdot m \cdot \log n)$, wobei l die Anzahl der Communitys ist, die entdeckt werden sollen; m und n sind jeweils die Anzahl der Kanten und Knoten. Die \tilde{O} -Notation verbirgt einen Faktor von maximal $(\log \log n)^2$.

Des weiteren haben wir die Performance unseres Algorithmus mit Hilfe des LFR Benchmark von Lancichinetti et al. bewertet. Wir haben herausgefunden, dass unser Algorithmus mit ausreichend Seed Knoten in der Lage ist Communitys sinnvoll zu erkennen.

2 Introduction

The modern study of networks tries to understand and extract information from complex networks. It plays a great role in many fields of science, such as biology, sociology and computer science. One important goal is the detection of community structure. A network is said to have community structure if its nodes can be separated into sets which are densely connected inside and share only few connections to other sets. These sets are called communities. Communities operate mostly independent of the rest of the network and can be analyzed as self-contained entity. Also, the interactions between communities describe a meta-network which reveals additional information about the network as a whole.

Non-overlapping community detection assigns each node to exactly one community [13] [15]. In contrast to that, *overlapping community detection* allows nodes to belong to multiple communities [6] [14]. Community detection may either be *crisp* or *fuzzy* [7]. For crisp detection it is a binary decision whether a node belongs to a community or not. Fuzzy detection, however, allows nodes to partially belong to (multiple) communities, often indicated by a *belonging factor* ranging between 0 and 1.

In section 3 we present current methods for non-overlapping and overlapping detection, including Newman’s modularity, as well as clique percolation and label propagation. In section 4 we discuss the fundamentals needed to understand our model for community detection. These include absorbing Markov chains (section 4.1, 4.2), random walks (section 4.3) and symmetric diagonally dominant linear systems (section 4.4). In section 5 we present our model for community detection. Finally, in section 6 we evaluate it based on the LFR benchmark proposed by Lancichinetti et al. [11], which uses random graph models to simulate complex networks.

2.1 Notation

In this section we introduce the notation used throughout this thesis. If M is a matrix then M_{ij} denotes the entry in the i th row and j th column of M . Furthermore, M_{i*} and M_{*j} describe the i th row and the j th column, respectively. Given a graph $G = (V, E)$, V is a set of n nodes and E is a set of m edges. The degree of a vertex v is denoted by $\deg(v)$. The adjacency matrix of G is defined as $A \in \mathbb{R}^{n \times n}$ with

$$A_{ij} := \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

3 Previous Work

This section serves as a quick overview to some algorithms for overlapping and non-overlapping community detection. As the field is very diverse, an extensive review is beyond the scope of this thesis. For more information, we recommend [21] and [5], two detailed surveys which have been a great source of information for this section.

3.1 Non-Overlapping Community Detection

The problem of community detection is not as well defined as it might seem. It is hard to define a community in a mathematically strict way. There are many valid definitions and often they differ greatly. In many cases some sort of arbitrariness and common sense is involved. One central property most definitions have in common is the assumption that there are many edges between members of a community and few edges between members of different communities.

One straightforward definition of a community would be a *clique*, i.e., a fully connected subgraph. Then the problem of community detection transforms to the well known problem of finding cliques within a graph. This might be a bit too restrictive though, as it is easy to come up with examples of groups in social networks which one would consider a community, but in which not everybody knows everybody.

An alternative approach would be to compare the number of connections within the community to the number of connections to the rest of the graph. A subgraph is said to be a *strong community* if for each vertex the number of edges to vertices within the community exceeds the number of edges to non-community vertices. In a *weak community* the total number of edges within a community exceeds the total number of outgoing edges. Radicchi et al. [15] used these concepts in their algorithm for community detection.

Another concept is the detection of communities via a *quality function*. A quality function is a function which rates partitions of a graph: If a partition reveals meaningful community structure it shall be given a higher score. Community detection then is reduced to finding a partition which maximizes the function.

One of the most important examples of a quality function is the *modularity* proposed by Newman et al. [13]. Modularity has been studied extensively and is a widely agreed upon measure for community structure up to today. It evaluates the goodness of a partition of a network into subgraphs by comparing it to a so called *null model*.

The null model describes a random graph where edges of the original graph are rewired at random, but vertices keep their degree. It assigns to each pair of vertices a probability that there is an edge between them.

$$\Pr(\text{edge between vertex } v \text{ and } w) = \frac{k_v \cdot k_w}{2m}$$

where k_v and k_w are the degree of vertices v and w and m is the total number of edges within the network.

A graph partition has high modularity (i.e., reveals community structure) if the actual number of edges within each subgraph exceeds the expected number of edges within each subgraph after edges were rewired according to the null model. The modularity Q of a graph is defined as

$$Q = \frac{1}{2m} \sum_{v,w} \left(A_{vw} - \frac{k_v \cdot k_w}{2m} \right) \delta(c_v, c_w)$$

where A is the adjacency matrix of the network and $\delta(c_v, c_w)$ is 1 if vertex v and w belong to the same subgraph, otherwise 0.

However, finding a partition which maximizes the modularity is an NP-hard problem [1], thus the optimal solution usually is infeasible to find. Approximate solutions can be obtained by finding eigenvectors in specially crafted matrices [12]. An in depth discussion on modularity and optimization algorithms can be found in [12] and [13]. For more information on non-overlapping community detection we recommend the survey by Fortunato [5].

3.2 Overlapping Community Detection

The traditional approach of non-overlapping detection assigns each vertex to exactly one community; however, this does not always model the real world. In a social network a person may belong to multiple communities (e.g., family, co-workers, sports club). Overlapping community detection takes this into account by assigning each vertex to one or more communities. There are many different approaches to overlapping community detection, two of which are presented in this section.

Clique Percolation The clique percolation method (CPM) proposed by Palla et al. [14] builds communities based on *k-cliques* (cliques of size k). Two k -cliques are considered adjacent if they share $k - 1$ vertices. Communities are identified as unions over all k -cliques that can be reached from each other through a series of adjacent k -cliques. They directly correspond to the connected components in a graph of all k -cliques, where two k -cliques are connected by an edge if they are adjacent.

Since vertices may belong to multiple k -cliques, this definition allows for overlapping communities. Small values for k (between 3 and 6) have been shown to give good results [14]. CFinder¹ is an implementation of this algorithm.

Label Propagation The underlying idea of label propagation is that vertices adopt the label of its neighboring vertices and form communities based on their label. Label propagation has been used for both non-overlapping [16] [22] and overlapping [6] community detection.

In the initialization phase of the algorithm each vertex is assigned a unique label. Then each vertex adopts the label which occurs most frequently within the set of labels of neighboring vertices. Ties are broken at random. This step is repeated until the vertices have found a consensus on their label. In the end, all vertices carrying the same label are identified with the same community. Label updates can happen synchronously (the label of a vertex at update step $i + 1$ depends on its neighboring set at step i) or asynchronously (vertices update their label in some fixed order).

The COPRA² algorithm by Gregory et al.[6] extends this idea to overlapping community detection. In this algorithm a vertex has a list of labels with corresponding belonging factors between 0 and 1. In the update step each vertex averages the belonging factors of its neighboring vertices and drops labels whose belonging factor is below some threshold.

This section should give a rough idea how diverse community detection algorithms can be. An extensive overview over a total of 14 current algorithms for overlapping detection, as well as evaluations and benchmarks can be found in [21].

¹<http://cfinder.org>

²<http://www.cs.bris.ac.uk/~steve/networks/software/copra.html>

4 Fundamentals

In this section we introduce absorbing Markov chains and prove some of their main properties. Furthermore, we discuss random walks, which are an application of Markov chains and a fundamental building block of our model. At last, we give a brief overview of near-linear-time solvers for symmetric diagonally dominant linear systems. The use of such solvers greatly improves the time complexity of our algorithm.

4.1 Markov Chains

Markov chains are used to model stochastic processes which change over time. The initial status of a process is known and the status of the process as time progresses is of interest.

Formally, a Markov chain is a sequence of random Variables X_1, X_2, X_3, \dots which fulfill the Markov property, namely that the next step only depends on the current step, i.e., $\Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \dots, X_n = x_n) = \Pr(X_{n+1} = x | X_n = x_n)$.

The set of possible values for X_i is called the state space of the chain. We assume a finite state space $S = \{s_1, \dots, s_n\}$. Markov chains are defined by the transition probabilities between these states, which can be represented in graph or matrix form. An example of both representations of a specific Markov chain can be found in figure 1 and 2.

Probability Distribution The state of a Markov chain at a given time is described by a probability distribution $\pi = (\pi(1), \dots, \pi(n))^T$, where $\pi(i)$ denotes the probability of being at state s_i . Every probability distribution satisfies:

$$\pi(i) \geq 0 \text{ for } 1 \leq i \leq n$$

and

$$\sum_{i=1}^n \pi(i) = 1$$

Stochastic Matrix Transitions in a Markov chain lead from one probability distribution to the next and happen in discrete steps. Each state has a certain probability to transition to another state. These probabilities can be expressed by a stochastic matrix P , where P_{ij} denotes the probability to reach state s_j from state s_i . A transition to the next probability distribution

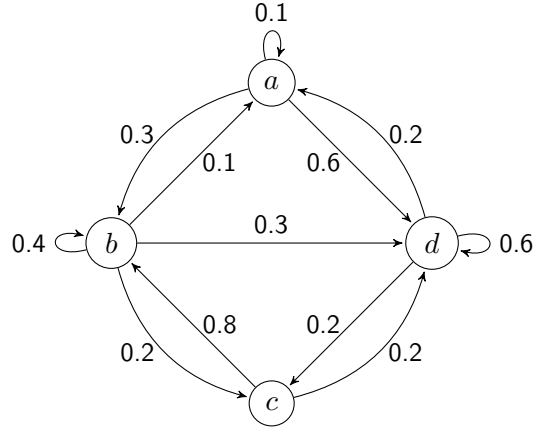


Figure 1: Graph representation of a Markov chain

$$P = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 0.1 & 0.3 & 0 & 0.6 \\ 0.1 & 0.4 & 0.2 & 0.3 \\ 0 & 0.8 & 0 & 0.2 \\ 0.2 & 0 & 0.2 & 0.6 \end{pmatrix} \end{matrix}$$

Figure 2: Transition matrix of the same Markov chain

can now be expressed by multiplying with P from the right: $\pi_{i+1} = \pi_i P$. P is a square matrix with

$$P_{ij} \geq 0 \text{ for } 1 \leq i, j \leq n$$

and

$$\sum_{j=1}^n P_{ij} = 1 \text{ for } 1 \leq i \leq n.$$

A Markov process needs an initial distribution π_0 . Since $\pi_{i+1} = \pi_i P$, the probability distribution after multiple steps can be expressed with powers of P :

$$\pi_k = \pi_{k-1} P = \pi_{k-2} P^2 = \cdots = \pi_0 P^k$$

Furthermore, the probability to reach state s_j from state s_i in k steps equals the ij th entry in P^k .

4.2 Absorbing Markov Chains

Absorbing Markov chains are special Markov chains with each state either being absorbing or transient. The probability of leaving an absorbing state is zero. Transient states have a nonzero probability to reach at least one absorbing state after a finite number of steps.

The stochastic matrix of an absorbing Markov chain with σ absorbing states $\{s_1, \dots, s_\sigma\}$ and τ transient states $\{t_1, \dots, t_\tau\}$ can be written as:

$$P = \left(\begin{array}{c|c} Q & R \\ \hline 0 & I \end{array} \right)$$

Where

- $Q \in \mathbb{R}^{\tau \times \tau}$, Q_{ij} denotes the probability to go from transient state t_i to transient state t_j .
- $R \in \mathbb{R}^{\tau \times \sigma}$, R_{ij} denotes the probability to go from transient state t_i to absorbing state s_j .
- $0 \in \mathbb{R}^{\sigma \times \tau}$ is the zero matrix.
- $I \in \mathbb{R}^{\sigma \times \sigma}$ is the identity matrix.

Using simple matrix transformations, it can be shown that:

$$P^k = \left(\begin{array}{c|c} Q^k & \sum_{i=0}^{k-1} Q^i R \\ \hline 0 & I \end{array} \right) \quad (1)$$

An important property of an absorbing Markov chain is that it always converges to a steady probability distribution. For a general Markov chain this is not the case. The final result of the community detection algorithm presented in section 5 will be a steady probability distribution of an absorbing Markov chain, so the following theorem is crucial.

Definition 1. $P^\infty := \lim_{k \rightarrow \infty} P^k$ is the transition matrix of an absorbing Markov chain for an infinite number of steps.

Theorem 1. P^∞ is well defined and

$$P^\infty = \left(\begin{array}{c|c} 0 & (I - Q)^{-1} R \\ \hline 0 & I \end{array} \right)$$

Proof. According to equation (1), we need to show

$$\lim_{k \rightarrow \infty} Q^k = 0 \quad (2)$$

and

$$\sum_{k=0}^{\infty} Q^k R = (I - Q)^{-1} R. \quad (3)$$

(3) is a direct consequence of (2) because the geometric series converges if the geometric sequence converges. Since Q is non-negative, a sufficient criterion for convergence of the sequence (2) is the existence of an l so that for all $1 \leq i \leq \tau$:

$$\sum_{j=1}^{\tau} [Q^l]_{ij} < 1$$

Because P is an absorbing Markov chain, we can find an l so that for each transient state t_i there is a non-zero probability p_i to reach an absorbing state after l steps. P^l is stochastic, so for all $1 \leq i \leq \tau$:

$$\sum_{j=1}^{\tau} [Q^l]_{ij} = 1 - p_i < 1$$

□

P^{∞} is well defined and describes an infinite number of steps of an absorbing Markov process. Because of (2), the transition probabilities to transient states in P^{∞} are all zero, consequently each state will be absorbed with a probability of one.

This leads us to the final result of this section:

Corollary 1. *The probability that a transient state t_i is absorbed in an absorbing state s_j equals the ij th entry in $(I - Q)^{-1} R$.*

More information on absorbing Markov chains can be found in the chapter 11 of Grinstead and Snell's book *Introduction to Probability* [8].

4.3 Random Walks

Random walks are an application of Markov chains and our model presented in section 5 heavily relies on them. A random walk (v_0, v_1, v_2, \dots) in a graph $G = (V, E)$ is a path generated by a stochastic process. v_0 is the start vertex and v_{i+1} is chosen uniformly random among the adjacent vertices of v_i .

The random walk can be described by a Markov chain: The state space corresponds to the vertices in G and the transition matrix P is defined as

$$P_{ij} = \begin{cases} \frac{1}{\deg(v_i)}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise.} \end{cases}$$

The probability to be at vertex j after a random walk of k steps starting at vertex i equals the ij th entry in P^k .

4.4 Symmetric Diagonally Dominant Linear Systems

A linear system is a set of linear equations

$$Ax = b$$

where A is a matrix and b , x are vectors of appropriate size. A and b are known and a value for x which satisfies all equations is of interest. Countless problems from numerical mathematics, engineering and science can be reduced to solving linear systems.

Fast algorithms to solve such systems are of particular interest. A naive approach using Gaussian elimination on an $n \times n$ matrix takes running time $O(n^3)$. Given that in real world applications, matrices easily contain millions of entries, such a running time is impractical. However, if the matrix contains a certain structure this can be exploited and far better running times can be achieved.

An important matrix for graph theory is a graph's Laplacian.

Definition 2. *Given a graph G with n vertices, its Laplacian is defined as $L \in \mathbb{R}^{n \times n}$ with*

$$L_{ij} := \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Laplacians play a great role in many fields, including computer graphics or scientific computing. A graph's Laplacian reveals many interesting properties of a graph. For example, the multiplicity of the eigenvalue zero gives the number of connected components in a graph. The algorithm we present in section 5.2 will rely on solving linear systems which are almost Laplacian. Many applications of Laplacians can be found in Fan Chung's book *Spectral Graph Theory* [2].

Definition 3. An $n \times n$ matrix A is said to be symmetric diagonally dominant (SDD) if it is symmetric and for all $1 \leq i \leq n$:

$$|A_{ii}| \geq \sum_{1 \leq j \leq n, i \neq j} |A_{ij}|$$

A graph’s Laplacian is an SDD matrix. Spielman and Teng recently had a major breakthrough when they showed that SDD linear systems can be solved in near-linear time [17, 4, 18]. Spielman and Teng’s algorithm (the ST-solver) combines numerical mathematics and graph theory to iteratively produce a sequence of approximate solutions which converge to the exact solution. The performance of such an iterative system is measured in terms of the time taken to reduce an appropriately defined approximation error by a constant factor. The time complexity of the ST-solver was reported to be at least $O(m \log^{15} n)$, where m is the number of nonzero entries [10]. Koutis, Miller and Peng [9, 10] developed a simpler and faster algorithm for finding ε -approximate solutions to SDD systems in time $\tilde{O}(m \log n \log(1/\varepsilon))$, where the \tilde{O} -notation hides a factor that is at most $(\log \log n)^2$. A highly readable account on SDD systems is the monograph by Vishnoi [20]. We summarize the main result, which we use as a black-box.

Theorem 2. [10, 20] Given a system of linear equations $Ax = b$, where A is an SDD matrix, there exists an algorithm to compute \tilde{x} such that:

$$\|\tilde{x} - x\|_A \leq \varepsilon \|x\|_A,$$

where $\|y\|_A := \sqrt{y^T A y}$. The algorithm runs in time $\tilde{O}(m \cdot \log n \cdot \log(1/\varepsilon))$ time, where m is the number of non-zero entries in A . The \tilde{O} -notation hides a factor of at most $(\log \log n)^2$.

This ability to solve SDD linear systems (exp. Laplacians) fast has been used to obtain many nearly-linear-time algorithms for applications such as semi-supervised learning, image processing or web-spam detection [19]. Or as Erica Klarreich puts it in her article *Network Solutions*³: “A new breed of ultrafast computer algorithms offers computer scientists a novel tool to probe the structure of large networks.”

³<http://www.simonsfoundation.org/mathematics-and-physical-science/network-solutions>

5 Our Model for Community Detection

Now that we introduced all necessary fundamentals we can present our model for community detection. Our goal is to find multiple, possibly overlapping communities within a network.

The algorithm we present needs to know some of the members of each community we want to discover. These members are called *seed nodes* and may belong to multiple communities. They need to be selected by the user and are handed to the algorithm as part of its input. The algorithm then uses random walks to extend the partial community-information given by the seed nodes to the rest of the network.

One specific application would be the detection of the political leanings in a social network like Facebook. The underlying assumption is that people are more likely to interact with people who have the same political beliefs. If one knows for some members of the network if they are either conservative or liberal but has no information about the rest of the network one may use the algorithm to decide for each member of the network if he or she is rather conservative or liberal.

The need for extra information via seed nodes sets this method apart from most other methods. This can be a disadvantage, as for many networks this information may not be present. But if a proper set of seed nodes can be specified our algorithm is quite flexible. By choosing a certain set of seed nodes the user can guide the algorithm to extracting very specific communities.

We say a node has *high affinity* to a community if it belongs to it and *low affinity* if it does not. Intermediate affinity values are possible and correspond to a partial belonging. The affinity of all seed nodes needs to be known to the algorithm beforehand. For all other nodes, the *non-seed nodes*, we want deduce the affinity to each community. We will use information given by the seed node's affinity and the network structure.

The fundamental idea is that non-seed nodes should adopt the affinities of seed nodes within their close proximity. But how do we define proximity? A naive approach would be to pick the seed node with the shortest path distance and adopt its affinities, but this does not model community structure very well: Consider a social network where people correspond to nodes and edges correspond to social interaction. Two people who do not know each other may have one friend in common or they may have several friends in common. In both cases the shortest path distance would be 2, but several common friends would be a much stronger indicator that these two people belong to the same community.

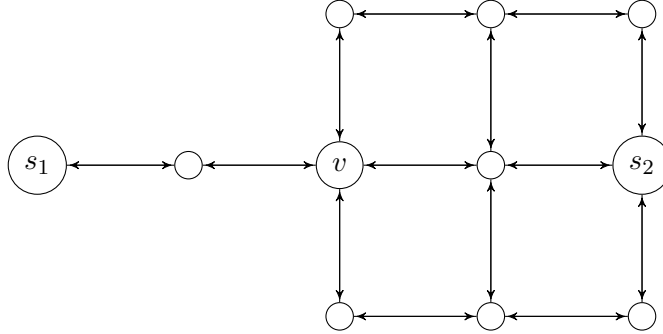


Figure 3: Example: Let $s_1 \in C_1$ and $s_2 \in C_2$ be seed nodes which belong to two implicit communities C_1 and C_2 . All other nodes are non-seed nodes. (s_1, v) and (s_2, v) both have a shortest path distance of 2. v apparently is more likely to be in the same community as s_2 than s_1 . It can be shown that a random walk starting at v reaches s_1 with a probability of $1/3$ and s_2 with a probability of $2/3$. As a result, v has an affinity of $1/3$ to C_1 and $2/3$ to C_2 .

This is why we use a different approach: We define a proximity measure based on random walks. The random walk starts at a non-seed node, traverses through the graph, and ends as soon as it reaches a seed node. The non-seed node then adopts the affinities of the seed nodes the random walk is likely to reach first.

Let us assume that a network, indeed, has community structure and contains some hidden communities C_1, \dots, C_l which we want to discover. For all seed nodes we know which of the communities they belong to. A defining property of a community is that it has many inner edges and few leaving edges, so a random walk starting a node within a community C_i should have a relatively high probability of staying within this community. This means that the probability that a random walk starting at a non-seed node reaches a seed node in C_i should be higher if the non-seed node lies within C_i than if it lies outside. After all, a non-seed node should adopt the affinities of seed nodes within the same community and thus be assigned correctly. As a result, the communities we detect should be conform with the community structure of the graph and the seed nodes. Figure 3 shows the advantage of random walks over the shortest path distance in a small example.

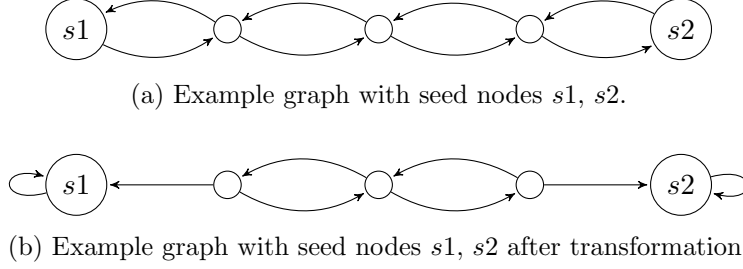


Figure 4: Remove outgoing edges and add self-loop for all seed nodes in an example graph. A random walk reaching $s1$ or $s2$ will stay there forever.

5.1 Single Community Detection

In this subsection we formally define our model for finding a single community. Later in section 5.3, we extend this model to multiple communities.

The input to our problem is an undirected, connected graph $G = (V, E)$ and a set of seed nodes $\emptyset \neq S \subset V$. We want to detect the community C . For all seed nodes $s \in S$ the affinity to C is part of the input. It is called $\beta(s)$ and may range between 0 and 1. $\beta(s) = 0$ means, that s does not belong to C and $\beta(s) = 1$ means that s belongs to C . Intermediate values are possible and correspond to a partial belonging. The algorithm returns the community by assigning an affinity $\beta(v)$ to all $v \in V \setminus S$.

Since random walks should end as soon as they reach a seed node, we transform G into a new graph G' as follows: First, we make the graph directed by replacing each undirected edge with two directed edges. Then for each seed node, we remove its outgoing edges and add a self-loop. The procedure is illustrated in figure 4. From now on, we only work with G' .

Theorem 3. *The random walks in G' define an absorbing Markov chain.*

Proof. Follows directly from the definition of absorbing Markov chains in section 4.2. There are no outgoing edges for seed nodes, hence, they represent absorbing states. For a non-seed node v let $s \in S$ be the seed node nearest to it. Since G is connected, there is one such node in S . In the undirected graph G' , there still is finite a path from v to s . After all, random walks in G' define an absorbing Markov chain with absorbing states S and transient states $V \setminus S$. \square

We want a non-seed node v to adopt the affinity of a seed node s if the probability that a random walk starting at v is absorbed at s is high. Let $E_{v \rightarrow s}$ be the event that an infinite random walk in G' starting at v is absorbed in s . According to Theorem 1, every infinite random walk will be absorbed and $\Pr(E_{v \rightarrow s})$ corresponds to an entry in P^∞ , where P is the transition matrix for random walks in G' , so $\Pr(E_{v \rightarrow s})$ is well defined and $\sum_{s \in S} \Pr(E_{v \rightarrow s}) = 1$.

Definition 4. We define for all non-seed nodes $v \in V \setminus S$:

$$\beta(v) := \sum_{s \in S} \beta(s) \Pr(E_{v \rightarrow s})$$

Thus $\beta(v)$ is a convex combination of $\{\beta(s) | s \in S\}$ with weights assigned according to the probability that a random walk starting at v is absorbed at a certain seed node.

5.2 Algorithm

In this section we present an efficient algorithm which calculates $\beta(v)$ for all $v \in V \setminus S$.

Theorem 4. Given a graph $G = (V, E)$ and seed nodes S , each with an affinity β , there exists an algorithm which computes $\beta(v)$ for all $v \in V \setminus S$ with running time $\tilde{O}(m \cdot \log n)$, where $m := |E|$ and $n := |V|$. The \tilde{O} -notation hides a factor of at most $(\log \log n)^2$.

Proof. The input to the algorithm is a graph $G = (V, E)$, the seed nodes S along with their affinity β . Let $n := |V|$, $\sigma := |S|$. The seed nodes are labeled $S = \{s_1, \dots, s_\sigma\}$ and the non-seed nodes are labeled $V \setminus S = \{v_1, \dots, v_{n-\sigma}\}$. The stochastic matrix $P \in \mathbb{R}^{n \times n}$ can be written as:

$$P = \left(\begin{array}{c|c} Q & R \\ \hline 0 & I \end{array} \right)$$

with $Q \in \mathbb{R}^{n-\sigma \times n-\sigma}$, $R \in \mathbb{R}^{n-\sigma \times \sigma}$.

We want to calculate $\beta(v_i)$ for a non-seed node v_i . According to Corollary 1, $\Pr(E_{v_i \rightarrow s_j}) = B_{ij}$ with $B := (I - Q)^{-1}R$. This yields:

$$\beta(v_i) = \sum_{j=1}^{\sigma} \beta(s_j) B_{ij}$$

The values $\beta(v_1), \dots, \beta(v_{n-\sigma})$ can be expressed as a vector. The matrix multiplication is linear, so it can be pulled out of the sum.

$$\begin{aligned} \begin{pmatrix} \beta(v_1) \\ \vdots \\ \beta(v_{n-\sigma}) \end{pmatrix} &= \sum_{j=1}^{\sigma} \beta(s_j) B_{*j} \\ &= \sum_{j=1}^{\sigma} \beta(s_j) (I - Q)^{-1} R_{*j} \\ &= (I - Q)^{-1} \sum_{j=1}^{\sigma} \beta(s_j) R_{*j} \end{aligned}$$

After all, $\beta(v_1), \dots, \beta(v_{n-\sigma})$ can be obtained by solving the linear system

$$(I - Q)x = b \tag{4}$$

with

$$x := \begin{pmatrix} \beta(v_1) \\ \vdots \\ \beta(v_{n-\sigma}) \end{pmatrix}$$

and

$$b := \sum_{j=1}^{\sigma} \beta(s_j) R_{*j}$$

Now, $(I - Q)$ has full rank, is diagonally dominant, but not symmetric. As a next step we will modify both sides of the equation to get an almost Laplacian SDD system with the same solution, which can be solved efficiently.

P describes a random walk and is defined as

$$P_{ij} = \begin{cases} \frac{1}{\deg(v_i)}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{otherwise.} \end{cases}$$

Q is the $(n - \sigma) \times (n - \sigma)$ -submatrix of P which holds the transition probabilities from transient to transient vertices. We define A as the adjacency matrix of $G[v_1, \dots, v_{n-\sigma}]$, the subgraph induced by all transient vertices, and D as the diagonal matrix with D_{ii} being the degree of v_i in G . Then $Q = D^{-1}A$ and we can rewrite (4) as

$$(I - D^{-1}A)x = b$$

If we multiply with D from the left we get the equivalent system

$$(D - A)x = Db.$$

D and A both are symmetric and $\sum_{j=1}^{n-\sigma} A_{ij}$ equals the degree of vertex v_i in $G[v_1, \dots, v_{n-\sigma}]$ whereas D_{ii} holds the degree of v_i in G , so $(D - A)$ is an SDD matrix.

Db and $(D - A)$ can be constructed in liner time in the number of non-zero entries. As a result, the running time of the algorithm is dominated by the near-linear running time of the SDD-solver described in Theorem 2. \square

5.3 Multiple Overlapping Community Detection

Our model for finding a single community can be naturally extended to multiple overlapping communities. The input is an undirected, connected graph $G = (V, E)$ and a nonempty set of seed nodes $S \subset V$, each with an affinity β . We want to detect communities C_1, \dots, C_l , so β is an l -tuple $\beta = (\beta_1, \dots, \beta_l)$ where $0 \leq \beta_i(s) \leq 1$ describes the affinity of seed node s to community C_i .

Definition 5. *The graph G' , the random walk, and $E_{v \rightarrow s}$ are defined as before. We define for all non-seed nodes $v \in V \setminus S$:*

$$\beta(v) := (\beta_1(v), \dots, \beta_l(v))$$

where

$$\beta_i(v) := \sum_{s \in S} \beta_i(s) \Pr(E_{v \rightarrow s})$$

Theorem 5. *Given a graph $G = (V, E)$ with l communities, seed nodes S along with their affinity $\beta = (\beta_1, \dots, \beta_l)$, there exists an algorithm which computes $\beta(v)$ for all $v \in V \setminus S$ with running time $\tilde{O}(l \cdot m \cdot \log n)$, where $m := |E|$ and $n := |V|$. The \tilde{O} -notation hides a factor of at most $(\log \log n)^2$.*

Proof. The values $\beta_i(v_1), \dots, \beta_i(v_{n-\sigma})$, $1 \leq i \leq l$ can be calculated by the algorithm for single community detection. So the running time for finding l communities is l times the time needed to find a single one. \square

An interesting property is, that if the affinity of all seed nodes sum up to a certain value the affinities of non-seed nodes do as well. This way, one can use a model where each vertex has a summed affinity of 1, which is individually distributed among different communities.

Corollary 2. *If $\sum_{i=1}^l \beta_i(s) = c$ for all $s \in S$, then $\sum_{i=1}^l \beta_i(v) = c$ for all $v \in V$.*

Proof.

$$\begin{aligned}
 \sum_{i=1}^l \beta_i(v) &= \sum_{i=1}^l \sum_{s \in S} \beta_i(s) \Pr(E_{v \rightarrow s}) \\
 &= \sum_{s \in S} \sum_{i=1}^l \beta_i(s) \Pr(E_{v \rightarrow s}) \\
 &= \sum_{s \in S} c \Pr(E_{v \rightarrow s}) = c
 \end{aligned}$$

□

6 Experimental Results

In this section we use the LFR benchmark proposed by Lancichinetti et al. [11] to evaluate the quality of our model. Even though our model is capable of overlapping community detection, to keep things simple, we focus on non-overlapping community detection only.

6.1 Implementation

Our reference implementation is written in the C++ programming language. It uses the Boost Graph Library⁴ for graph manipulation and traversal, and the Eigen library⁵ to solve numerical problems. Since it is a very recent finding that SDD systems can be solved in near-linear time, there is no stable implementation of this fast algorithm yet. Instead, we use a direct sparse Cholesky decomposition⁶. This algorithm does not have a near-linear time complexity, but has been optimized for real world applications and performs quite well in practice. In our example the detection of 200 communities in a graph consisting of 10000 vertices and 150000 edges on a consumer laptop (Intel Core i3, 4GB RAM) takes approximately 30 seconds.

6.2 Benchmark

The LFR benchmark is used widely for evaluation of community detection algorithms [21] [11]. Node degrees and community sizes of many real world networks follow a *power law distribution* [3] and the LFR benchmark aims to simulate these kinds of networks. A value x is said to obey a power law distribution if it occurs with a probability $p(x) \propto x^{-\alpha}$ for a constant parameter α . Usually, α ranges between 2 and 3. This means low values for x are common and high values are rare.

⁴http://www.boost.org/doc/libs/1_55_0/libs/graph

⁵<http://eigen.tuxfamily.org/>

⁶http://eigen.tuxfamily.org/dox-devel/group__SparseCholesky__Module.html

The LFR benchmark generates a random graph and assigns each vertex to exactly one community. This process can be controlled by the following parameters:

- N controls the number of nodes in a graph.
- $\langle k \rangle$ controls the average node degree.
- γ and β control the exponent of the power law distribution of node degree and the community size, respectively.
- μ is called the *mixing parameter* and controls how interweaved the communities are. Each node shares a fraction $1 - \mu$ of its edges with other nodes of the same community and a fraction of μ with nodes that belong to a different community.

An illustration of a graph generated by the LFR benchmark is shown in figure 5.

Setup As the first step in our test setup, the LFR benchmark generates a graph $G = (V, E)$ consisting of communities C_1, \dots, C_l and assigns each vertex to one community C_i . Then a random subset $S \subset V$ is chosen as a set of seed nodes. For each seed node $s \in S$ $\beta(s) = (\beta_1(s), \dots, \beta_l(s))$ is chosen with:

$$\beta_i(s) = \begin{cases} 1 & \text{if vertex } s \text{ belongs to community } C_i \\ 0 & \text{otherwise} \end{cases}$$

The fraction of seed nodes is controlled by parameter σ so that $|S| = \sigma|V|$. The number of communities in the LFR-graph appears to be linear in the size of graph, so for a fixed σ the number of seed nodes remains constant for different graph sizes.

Next, community detection is performed. The multiple community detection algorithm from section 5.3 returns for each non-seed node $v \in V \setminus S$ and affinity vector $\beta(v) = (\beta_1(v), \dots, \beta_l(v))$. We assign vertex v to the community with the greatest affinity value, i.e., v is assigned to C_i where $i = \operatorname{argmax}_{1 \leq i \leq l} \beta_i(v)$. We call the original community of vertex v chosen by the LFR benchmark $c(v)$ and the community returned by the algorithm $\hat{c}(v)$. As an intuitive quality measure Q , we use the fraction of correctly assigned vertices. Q ranges between 0 (bad) and 1 (good).

$$Q = \frac{|\{v \in V | c(v) = \hat{c}(v)\}|}{|V|}$$

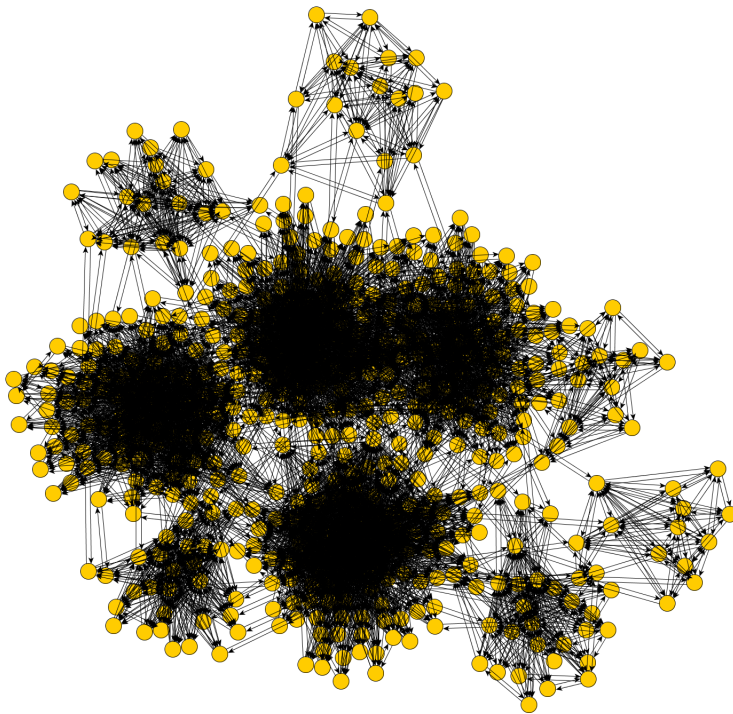


Figure 5: A graph generated by the LFR benchmark with $N = 500$, exponents $\gamma = 2$ and $\beta = 2$, average node degree $\langle k \rangle = 20$, and mixing parameter $\mu = 0.05$. It consists of 10 communities.

We calculate this quality measure for our community detection algorithm on various graphs. Different values for the parameters γ , β , $\langle k \rangle$, σ and μ are chosen to simulate a wide range of scenarios. For each setting 100 runs were performed and the average result for Q was taken. The result is shown in figure 6. All benchmark-parameters are again summarized in figure 7.

6.3 Evaluation

The benchmark setup we used to evaluate our model heavily relies on the methods proposed and used by Lancichinetti et al. In their paper *Community detection algorithms: a comparative analysis* [11] they employ the LFR benchmark to evaluate modern methods for overlapping and non-overlapping community detection. They use the less intuitive *Normalized Mutual Information* as quality measure and slightly different parameters for graph generation, so one has to be careful when comparing our results

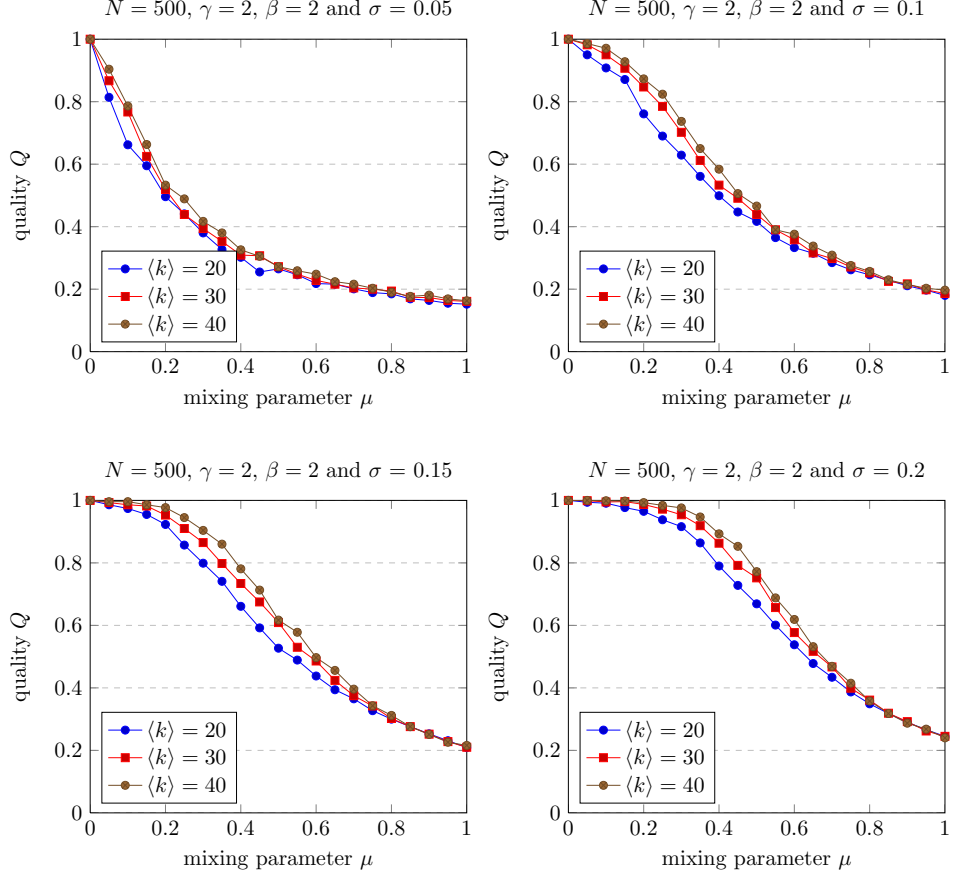


Figure 6: Results of the benchmark. The plotted value for Q is the average over 100 runs with different random seed vertices and graphs generated for the set of parameters.

γ	Exponent of power law distribution of node degree
β	Exponent of power law distribution of community size
$\langle k \rangle$	average node degree
N	Number of nodes
σ	Fraction of nodes that are seed nodes
μ	Mixing parameter
Q	quality measure

Figure 7: Summary of all parameters for the benchmark

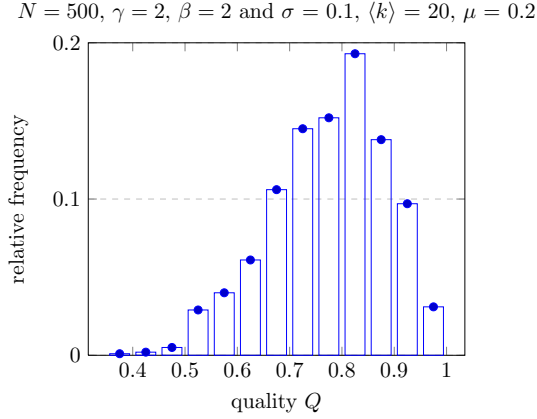


Figure 8: Histogram over the distribution of Q . 1000 data points created by 1000 runs on the same graph. The seed nodes were chosen randomly for each iteration.

with theirs, however, their paper may help interpreting our results. Further studies should strive to recreate their test setup to compare our method to state-of-the-art methods.

The results of the benchmark in figure 7 show that the performance of our model greatly depends on the choice of seed nodes. The more seed nodes are available, the better is the quality of the output. Given a network with mixing parameter $\mu = 0.3$ the algorithm fails when given only 5% seed nodes ($Q \approx 0.4$) but works fine when given 20% seed nodes ($Q \approx 0.95$).

There is also a correlation between the mixing parameter of the graph and the fraction of seed nodes needed to archive an accurate detection. The more diffuse the communities are, the more seed nodes are needed for good detection. To archive a quality of $Q = 0.95$ in a graph with mixing parameter $\mu = 0.1$ a fraction of 10% seed nodes are needed. If $\mu = 0.3$ it takes 20% seed nodes to archive the same quality. However, if the mixing parameter of the graph becomes too big ($\mu > 0.5$) the algorithm fails even for large fractions of seed nodes. Also high average vertex degrees slightly improve the quality of the detection.

Furthermore, figure 8 shows that for the same graph and different sets of seed nodes of the same size the quality of the detection varies a lot. So the choice of good seed nodes is critical for the performance of the algorithm. It could be an interesting subject for further studies to identify criteria for the choice of good seed nodes.

7 Conclusion

The goal of this work was the development and evaluation of an algorithm for overlapping, fuzzy community detection based on random walks. The algorithm extracts specific communities from a network based on a set of seed nodes. It runs in time near-linear in the number of communities times the number of edges in the network. We used the LFR benchmark for evaluation and found that, given a good set of seed nodes, the algorithm is able to correctly reconstruct the communities of a network.

The number of communities contributes linearly to the algorithm’s running time. This means that in a scenario where the number of communities is proportional to the number of nodes, the algorithm’s running time would be squared in the number of nodes, thus may not be efficient. It may be better suited for scenarios with only a low number of communities, such as the detection of political leanings of users of a social network.

Further work should focus on a rigorous analysis of the proposed method based on the LFR benchmark. One should aim to evaluate the performance under different settings, as well as to compare the algorithm to other methods for community detection. It could also be of interest to test the usefulness of our model in real world situations or to analyze different strategies for choosing good seed nodes.

References

- [1] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing Modularity is hard. *ArXiv Physics e-prints*, August 2006.
- [2] Fan R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, December 1996.
- [3] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.
- [4] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 494–503. ACM, 2005.
- [5] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, February 2010.
- [6] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, October 2010.
- [7] S. Gregory. Fuzzy overlapping communities in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2:17, February 2011.
- [8] Charles M. Grinstead and Laurie J. Snell. *Grinstead and Snell’s Introduction to Probability*. American Mathematical Society, version dated 4 july 2006 edition, 2006.
- [9] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, pages 235–244. IEEE Computer Society, 2010.
- [10] I. Koutis, G. L. Miller, and R. Peng. A nearly $O(m \log n)$ -time solver for SDD linear systems. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, pages 590–598. IEEE Computer Society, 2011.
- [11] A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physics Review E*, 80(5):056117, November 2009.

- [12] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physics Review E*, 74(3):036104, September 2006.
- [13] M. E. J. Newman. From the Cover: Modularity and community structure in networks. *Proceedings of the National Academy of Science*, 103:8577–8582, June 2006.
- [14] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, June 2005.
- [15] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Science*, 101:2658–2663, March 2004.
- [16] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physics Review E*, 76(3):036106, September 2007.
- [17] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2004.
- [18] D. A. Spielman and S.-H. Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. Available at: <http://arxiv.org/abs/cs/0607105>. Submitted to SIMAX, 2008.
- [19] Shang-Hua Teng. The laplacian paradigm: Emerging algorithms for massive graphs. In Jan Kratochvíl, Angsheng Li, Jiří Fiala, and Petr Kolman, editors, *Theory and Applications of Models of Computation*, volume 6108 of *Lecture Notes in Computer Science*, pages 2–14. Springer Berlin Heidelberg, 2010.
- [20] N. K. Vishnoi. $Lx = b$. *Laplacian Solvers and their Algorithmic Applications*. Now Publishers, 2013.
- [21] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping Community Detection in Networks: the State of the Art and Comparative Study. *ArXiv e-prints*, October 2011.

- [22] J. Xie and B. K. Szymanski. LabelRank: A Stabilized Label Propagation Algorithm for Community Detection in Networks. *ArXiv e-prints*, March 2013.